Managing Variability in Workflow with Feature Model Composition Operators

Mathieu Acher, Philippe Collet, Philippe Lahire, Robert B. France

Managing Variability in Workflow with Feature Model Composition Operators

Mathieu Acher[1], Philippe Collet[1], Philippe Lahire[1], and Robert France[2]

[1] University of Nice Sophia Antipolis, France
I3S Laboratory (CNRS UMR 6070),
06903 Sophia Antipolis Cedex, France
{acher,collet,lahire}@i3s.unice.fr
[2] Computer Science Department,
Colorado State University, USA
france@cs.colostate.edu

Abstract. In grid-based scientific applications, building a workflow essentially involves composing parameterized services describing *families of services* and then configuring the resulting workflow product line. In domains (e.g., medical imaging) in which many different kinds of highly parameterized services exist, there is a strong need to manage variabilities so that scientists can more easily configure and compose services with consistency guarantees. In this paper, we propose an approach in which variable points in services are described with several separate feature models, so that families of workflow can be defined as compositions of feature models. A compositional technique then allows reasoning about the compatibility between connected services to ensure consistency of an entire workflow, while supporting automatic propagation of variability choices when configuring services.

1 Introduction

In grid-based scientific collaboration communities, scientists build workflows by assembling services that, in many cases, perform complex tasks [1]. For example, in the grid-based medical imaging community, scientists compose diverse image processing services to create chains that meet their specific needs. To support reuse, services can be parameterized, thus allowing a scientist to tailor a service to a particular context. Current approaches to assembling grid-based services are labour-intensive [2] and require scientists to manually manage knowledge about *i)* the variable points supported by services, and *ii)* the restrictions on how services must be tailored and composed to meet end-to-end Quality of Service (QoS) or other requirements. When a wide variety of parameterized services exists, the tasks of identifying, tailoring and composing services become tedious and error-prone [3], especially in grid-based medical imaging. As identified in previous work [4, 5], the difficulty of provisioning and composing such services stems from the lack of mechanisms for managing variabilities within and across services. The above problems give rise to the following challenges. The first challenge is to provide mechanisms that enable service providers (e.g., research scientists, workflow or grid experts) to capture the commonalities and variabilities in parameterized services that are offered on the grid. The second challenge is concerned with providing support for tailoring and composing services such that service consumers can ensure the consistency of resulting workflows with well-defined properties.

In order to meet these challenges, we describe in this paper a rigorous approach to composing parameterized services into workflows. The approach utilizes Software Product Line (SPL) and Aspect-Oriented Modeling (AOM) techniques. The goal of SPL engineering is to produce reusable artifacts that can be used to efficiently build members of a software product family [6]. The reusable artifacts encapsulate common and variable aspects of a family of software systems in a manner that facilitates planned and systematic reuse. A parameterized grid-based service can be viewed as an SPL. We observe that the variabilities in a parameterized service can be described along a variety of dimensions. For example, in a medical imaging service, three commonly used dimensions concern QoS features, image formats and communication protocols for data transmission. We rely on prior results [7] to separate aspect models which exhibit variability and compose them to produce a comprehensive variability model. This modular technique allows applying separation of concerns principles and thus limits the considered variabilities only to relevant concerns.

In the present approach, a workflow is created by first composing *families of services* and then configuring the resulting workflow product line. Feature models [8,9,10] (FMs) are used to describe the common and variable features in a tailorable service. The variable points in a parameterized service are described by multiple separate FMs, where each FM describes a set of variable points in a particular dimension. A set of composition operators is used to *i)* insert a concern with variability into the description of services and *ii)* merge variability models of connected services. The FM composition operators are used to reason, at the FM level, on services' dependencies specified by the user and identified as active in the workflow. Using these operators, it is possible *i)* to analyze the entire workflow by checking the consistency of families of dependent services and *ii)* to infer variability information and propagate user choices according to variability information described in each family of services. We also consider the impact that workflow constructs (sequence, concurrency, if-then-else condition) have on service composition. The approach assists users with their decision-making process and can largely reduce the sets of configurations to be considered when tailoring and composing services.

2 Motivation and Overview of the Approach

Scientific workflows are increasingly used for the integration of existing, legacy tools and algorithms to form larger and more complex applications, e.g., for scientific data analysis or computational science experiments. In the medical imaging area (e.g., see [11]), scientific workflows are deployed on grids for addressing the computing and storage needs arising from manipulation of large fragmented medical data sets on wide area networks. Service-oriented architectures (SOA) are especially suited to such a domain: There is a need for *reusable self-contained services* that provide standardized interfaces for calling application code as well as information exchange protocols [12]. In SOA, services are atomic entities that are composed to produce complex (business) processes implementing workflows.

Managing Many Concerns. Many scientific services have a large number of input ports and parameters, but not exclusively. Individual data items processed

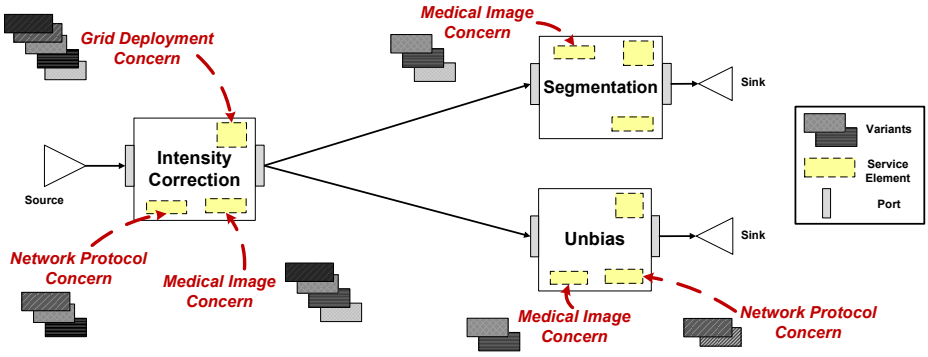


Fig. 1. Workflow, Services, Concerns and Variants

by scientific workflows are very much related to each other and there is a need to maintain data cohesion: Dependencies between different services within a workflow system must be managed from several perspectives. More generally, deployed services contain a lot of information related to the environment in which they are deployed and composed. In the case of medical imaging services on the grid, service providers supply basic imaging services, implemented in a variety of languages, packaged with information needed to compose the services with other services. In addition, providers have to manage the numerous non-functional properties that are exploited during deployment or runtime, in order to meet quality of service (QoS) goals. The overall issue for users of the workflow is to deal with services' dependencies in the workflow while addressing a large amount of concerns.

An SPL Approach. Rather than providing services in hopes that opportunities for reuse will arise during the design of a workflow, a proactive strategy is to apply SPL principles and plan which characteristics of a service are likely to be systematically reused. The ability to efficiently create many variations of a service and capitalize on its commonalities can improve its composability and increase the extent to which service logic is sufficiently generic so that it can be effectively reused. Our previous work indicates that there is significant *variability* in medical imaging services on the grid [4, 5]. For example, a service is able to read and process several medical image formats; some services use network protocols that do not support the transmission of a "receive acknowledge" to indicate that the packet has been received whereas some other services do have this capability. A medical imaging service that exhibits variability can thus be treated as an SPL or *family of services*. For each concern of a service, there are several alternatives that the user has to consider and choose from to derive an actual service. Adopting an SPL approach, a family of services is described from a variety of *variable concerns*, i.e., a concern with variation points, and thus can be represented as a set of variants.

We believe that separation of functional and non-functional concerns with variability can improve the reusability of services. In Figure 1, three concerns are woven into elements of the Intensity Correction service to augment its description. *Medical Image* describes the medical images input formats that the service is able to process. *Grid Deployment* provides information about service deployment on the grid, e.g., the operating system needed to run the service on

the grid. *Network Protocol* represents the network protocol used by the service. When the service Intensity Correction is connected to the services Segmentation and Unbias, several concerns of Intensity Correction may be related to several concerns of Segmentation and Unbias. For instance, users may require that the medical image output of Intensity Correction be compatible with the medical image input of Segmentation and Unbias; or that the network protocol used by Intensity Correction be consistent with the network protocol used by Unbias.

Key Issues. The goal of the SPL approach promoted in the paper is to manage not only the variability of the family of services but also the variability of the resulting composed services. The following key challenges are targeted by our approach. A first challenge is *to cope with multiple dimensions of a family of services* by providing mechanisms to augment the service description with variants from the concerns dimension. These variants can be woven in at several points (e.g., port, interface) in the service description. A second challenge is *to ensure that families of services are consistently composed in the workflow.* At the workflow level, the links between services and their semantics exist in various forms (complex dependencies, input/output dataflow, provided/required interfaces compatibility, etc.). A developer must identify and specify how concerns with variability are treated when services are composed. The actual selection of variants in a large and complex SPL can be a tedious and error-prone task [13]. A third challenge is *to assist the user in selecting the right variant* for each family of services and for each dimension. These choices should be consistent for the entire workflow and should not violate the specified restrictions on concerns.

3 Modeling Concerns with Variability in Workflow

A medical imaging service should have high *variability*, that is, a user should be able to efficiently extend, change, customize or configure the service in a particular context [13]. The medical image format provides an example: some formats can anonymize the medical image by removing all patients metadata; some can compress and/or reduce the image size while the format header may differ. Services often must address several concerns. In this paper, the term *concern* is used in a broad sense and may range from high-level requirements to low-level implementation issues, refer to measurable properties or to the behaviour of the system.

Separation of Concerns (SoC). The number of variants and choices for each concern of a service can be extremely large and can be a threat to scalability: variability descriptions quickly become too complex to manage, evolve or analyze by users. When modeling variants of a service, applying SoC principles and providing support to the modularisation of variability description can make them scale better. In our approach, the SoC is twofold. Firstly, concerns are associated with and described according to precisely defined elements of a service (e.g., Dataport). Secondly, we support the separation of variability models instead of the use of a large and monolithic variability model: The variability description of a service can be modularized, where each modular model focuses on a well-identified concern.

Modeling Workflow and Service. We first need to model what is a service, what its elements are and how services are assembled in workflows. Figure 2 shows a metamodel[3] that describes the form of services supported by our approach.

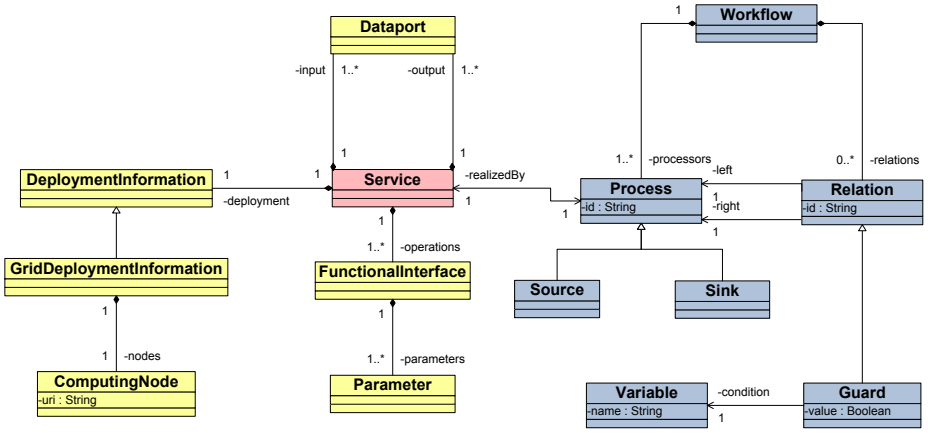


Fig. 2. Metamodel of Service and Workflow

A service describes data items in Dataports. Input dataports hold references to the data to be processed and output data ports contain references to the data produced by a service. FunctionalInterface represents the interfaces (legal operations with their parameters) exposed by a service. DeploymentInformation provides information about the deployment of the service, and a specialized GridDeploymentInformation references the ComputingNodes on which services are deployed. In addition, the metamodel describes how services can be connected in a workflow represented as a set of Processes. Two special processor nodes are also defined: Sources produce data to feed the workflow and Sinks collect the data produced. For the purpose of the paper, we consider that a Process is bound to one and only one Service in the sense that a service *realizes* a process. The connection between processes is specified as a partial ordering: The right part of Relation is a process that must wait for the end of the left part to start its own execution. If-then-else conditions can be expressed with a Guard which evaluates a predicate on objects of Variable.

Modeling Variability. A concern (e.g., *medical image format*) of a service can exhibit a set of variable points (e.g., alternatives, optionality). We choose to describe its variability with a Feature Model (FM). FMs are widely used to model a family (e.g., an SPL) in terms of common and variable features. Several definitions of *feature* appear in the literature, ranging from "anything users or client programs might want to control about a concept" [8] to "an increment in

[3] While there exists more comprehensive metamodels for service descriptions, we chose to use a simple metamodel that shows only the service concepts needed to understand our approach.

product functionality" [9]. These definitions indicate that FMs, like concerns, are not only relevant to requirement engineering but they can also be applied to design or implementation [14].

In Figure 3, a family of medical images is represented by a FM and has two mandatory features, ModalityAcquisition and Format, which imply that each valid configuration of a medical image should include these two features. An optional feature is Anonymized, which states whether all patients metadata of the medical image are included or not. There are also three alternatives of medical image format: Nifti, DICOM or Analyze features form a Xor-group. It means that at least and at most one feature must be selected. Finally, an MRI medical image has either the parameter T1 or T2 or both of them: T1 and T2 form an Or-group. A FM thus describes the set of valid feature combinations. Every member of a family is represented by a unique combination of features. For instance, each valid feature combination of a FM representing a family of requirements corresponds to an actual requirement. In the remainder of the paper, *a combination of selected features is called a configuration of a FM and is represented as a set of features.* In Figure 3, a valid configuration of the FM is as follows: {MedicalImage, ModalityAcquisition, Format, CT, DICOM}. A configuration is valid if all features contained in the configuration and the deselection of all other features are allowed by the semantics of FM [9, 10].

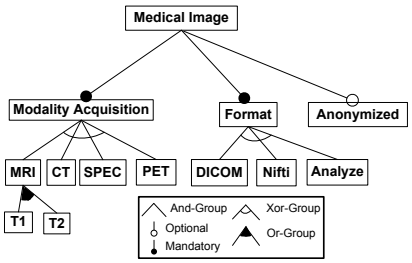


Fig. 3. Medical Image FM

Weaving Concern. Figure 4 describes how concerns which exhibit variability (called VariableConcerns) can be composed in a service description. We consider that a FM (resp. configuration) is an abstract view of a variable concern (resp. variant): A VariableConcern is described with a FeatureModel where each Configuration of a FeatureModel corresponds to a concrete Variant

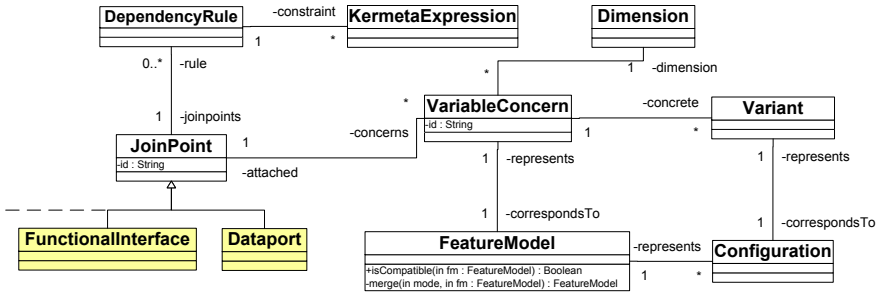


Fig. 4. Join Point Modeling

There is need to specify *where* the concern is inserted in the description of a service. A mechanism is required to weave a concern into a service model that

conforms to the service metamodel of Figure 2. We propose that each model element of the service metamodel (e.g., Dataport, FunctionalInterface, etc.) can inherit from JoinPoint. Join points represent well-defined places in the structure of a service where additional behaviour can be attached. In our case, a VariableConcern can be attached to any JoinPoint. For example, a concern dealing with the description of medical images supported by a service can be attached to the Dataport. Another concern can be attached to Dataport, e.g., to describe the security of data. It is also possible to describe how medical images will be stored on the grid. As a result, several concerns along several Dimensions can be associated to a JoinPoint. The actual weaving of a VariableConcern in a specific JoinPoint can be achieved using AOM composition approaches. In the rest of the paper, a dotted arrow which links a FM to a dashed border box means that we weave the FM into an instance of a JoinPoint. For example, in Figure 5 of Section 4, FM_{o1}, is woven to an actual output Dataport of $FService_1$. (Output Dataport is the name mentioned in the box and is a shortcut to name an instance of an output Dataport of a service.)

4 Reasoning on Workflow Concerns

Services are composed in the workflow while several concerns can be weaved into various elements of services. For some reasons, mainly due to the interconnection of services in the workflow, elements of services may be dependent. As a result, concerns attached to these elements may, in turn, be dependent on each other. This typically occurs when concerns belong to the same dimension.

Dependency Modeling. For instance, the medical image output format of a service S_1 is considered to be compatible with the medical image input format of another connected service S_2 in the workflow (see Figure 5). We need to express, at the model level, that an output Dataport of S_1 has to be compatible with an input Dataport of S_2 if they are to be connected. We define some DependencyRule(s), which are associated to JoinPoint elements of the service metamodel and that restrict in some way[4] the VariableConcerns.

The compatibility relation between two concerns vc_1 and vc_2 only considers concerns that belong to the same dimension. It is defined as follows: For at least one Variant of vc_1, there is an equal Variant in vc_2 (and vice-versa). The same relation applies to all dependency rules. In our implementation [15] of the approach, we formalize these rules using the Kermeta language [16] and the compatibility relation corresponds to the function $checkVariableConcerns$. **Rule 1** defines the compatibility between S_1 and S_2 Dataports as described above. It is expressed in Kermeta as follows:

```
s1.output.each{op |
    if op.connectInput.size > 0 then
        // some input ports (of service s2) are connected to op
        op.connectInput.each{ ip |
          // merge concerns of output port op and input port ip of s2
          checkVariableConcerns(op.concerns, ip.concerns)
        }
    end
}
```

[4] The use of constraints between FMs is not considered in the paper (see Section 6)

Another rule, **Rule 2**, states that when two services are connected, the concerns associated to their FunctionalInterfaces are to be compatible. If a service S_1 does not support HTTP whereas the service S_2 only supports HTTP, users can be prevented from an inconsistency of service S_1 and service S_2. It is also possible to define dependency rules between concerns *without* considering the connection between services in the workflow. For instance, **Rule 3** states that if services are deployed on an equal computing node (a resource on the grid), all concerns must be compatible with each other. The concern can refer to the set of operating systems in which the services can be deployed and executed. In this case, if a service S_1 can only run on the Linux operating system whereas another service S_2 can only run on the BSD operating system, the two services cannot be deployed on the same computing node.

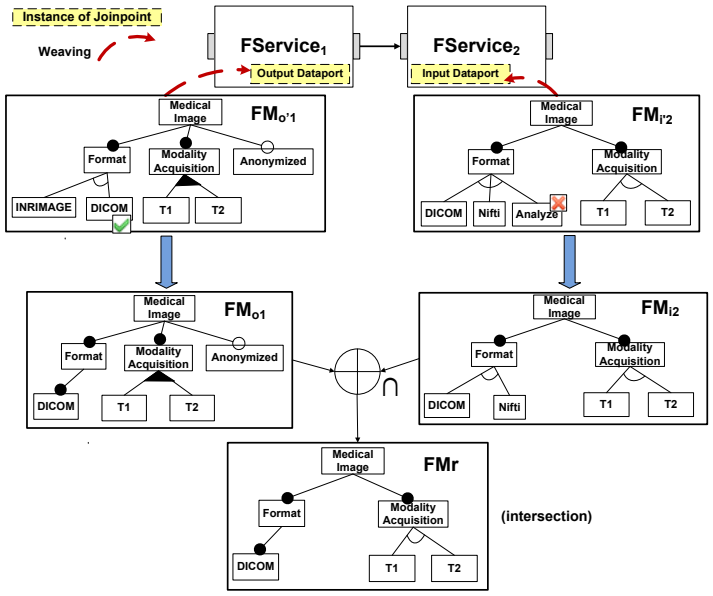


Fig. 5. Consistency checking concerns while shrinking variability choices

As a summary, DependencyRules express restrictions on the variants that can be derived from concerns attached to families of service elements. This boils down to ensuring the consistency of each set of Variant(s) associated with a Join Point of a Service. To implement the reasoning, we rely on our previous work, defining a set of composition operators for FMs. In [7], the semantics of each operator has been given in terms of the expressed configurations. Here, we focus on the merge operator which is dedicated to the composition of FMs that represent variable concerns along the same dimension.

Merge Operator. When two FMs share several features and are different viewpoints of a concern, the goal of the merge operator is to merge the overlapping parts of the two FMs to obtain an integrated model of the system. For example, users want to merge two medical image descriptions represented by two FMs,

FM_{I_2} and FM_{0_1}, depicted in Figure 5. The merge operator uses a name-based matching: two features match if and only if they have the same name. The merge process starts from the root features of FM_{I_2} and FM_{0_1}. Medical Image of FM_{I_2} and FM_{0_1} match. Then, when two features have been merged, the whole process proceeds with their children features. Two modes are defined for the merge operator. The *intersection* mode is the most restrictive option: The merged FM, FMr, expresses the common valid configurations of FM_{I_2} and FM_{0_1}. The *union* mode is the most conservative option: the merged FM, FMr, can express either valid configuration of FM_{I_2} or valid configuration of FM_{0_1}. The variability information associated to features in the merged FM is set according to the defined rules. These rules (see [7] for more details) are different according to the merge mode and the properties that one may want to preserve.

We now formalize some properties of the merged FM with respect to the sets of configurations of input FMs. Let f be a FM and $[\![f]\!]$ denotes its set of configurations. The relationship between a merged FM *Result* in intersection mode and two input FMs FM_1 and FM_2 is denoted $FM_1 \oplus_\cap FM_2 = Result$. It can be expressed in terms of sets of configurations:

$$[\![FM_1]\!] \bigcap [\![FM_2]\!] = [\![Result]\!] \qquad (M_1)$$

According to the example of Figure 5, a valid configuration of the merged FM, FMr, is valid in FM_{I_2} and in FM_{0_1} at the same time. The DICOM feature is always part of any valid configuration of FM_{I_2} and FM_{0_1} whereas the Nifti feature cannot be part of any valid configuration of FM_{0_1}. As a result, DICOM is a mandatory feature of the merged FM while the Nifti feature is not part of the merged FM. The following relation can be shown to hold: $[\![FMr]\!] = [\![FM_{I_2}]\!] \bigcap [\![FM_{0_1}]\!]$

In the union mode, we want to obtain a merged FM that represents the set of configurations of FM_{I_2} and FM_{0_1}. The merge operator in the union mode is denoted $FM_1 \oplus_\cup FM_2 = Result$. In the same way, we define the relationship between a merged FM *Result* and two input FMs FM_1 and FM_2 in terms of sets of configurations:

$$[\![FM_1]\!] \bigcup [\![FM_2]\!] = [\![Result]\!] \qquad (M_2)$$

Using FMs, the user can configure a family and thus derive an individual product (see Section 3). Observing that a FM in which there is no variability represents exactly one configuration, we decide to consider that a configuration of a FM *is* a FM. The rationale behind *considering configuration as an FM* is that it allows one to use the merge operator at each step of the configuration process.

Reasoning on Dependencies. In Figure 5, we focus on the medical image format concern. To illustrate our approach, we consider a very simple workflow where two processes are executed in sequence. $FService_1$ is connected to $FService_2$. FM_{O_1} (resp. FM_{I_2}) represents the medical image format information of $FService_1$ (resp. $FService_2$) and is associated to the output (resp. input) dataport. In this context **Rule 1** applies: The connection between $FService_1$ and $FService_2$ implies that FM_{O_1} and FM_{I_2} must be compatible. It is thus neces-

sary to check if the set of configurations of FM_{O_1} is equal or included in the set of configurations of FM_{I_2} (and vice versa). Our technique is to compute the merge in intersection mode between FM_{0_1} and FM_{I_2}. If the merged FM does not represent an empty set of configurations, then there must be at least one configuration that is valid in FM_{O_1} and FM_{I_2}. The consistency checking can thus be achieved. In the example, such an FM exists (see FMr).

The benefits of computing the merged model are threefold. (1) The restriction on the concerns shrinks the variability choices in FM_{O_1} and FM_{I_2}. This restriction is represented by the merged FM. In this case, there is no longer need to consider the Nifti feature in FM_{I_2} or Anonymized feature in FM_{O_1}. (2) The user can use the merged FM to configure FMs of $FService_1$ and $FService_2$ at a time. One configuration of the merged FM corresponds to the same configuration in $FService_1$ and $FService_2$. For example, if the user selects $T1$ in the merged FM, then it implies that the feature $T1$ associated to $FService_1$ and $FService_2$ are also selected. (3) The merged FM can be the basis for reasoning on the compatibility with another FM. Let us now consider that $FService_1$ is also connected to another service $FService_3$ (see Figure 7(a), Section 5). The output dataport of $FService_1$ is thus dependent on the input dataport of $FService_3$. As a result, the new restriction on FM_{O_1}, represented by the merged FM, should be used to reason on the compatibility between $FService_1$ and $FService_3$.

5 Consistent Workflow Configuration

5.1 Impact of Workflow Constructs

We have seen in Section 4 how we can reason on two services that are connected. Workflows usually have more than two services executed in sequence, and others with parallel computations and branching through if-then-else constructs. It is necessary to ensure the consistency of concerns configurations considering the various workflow constructs (e.g., sequence, concurrency, condition).

Sequence. Figure 6 shows three services $FService_1$, $FService_2$ and $FService_3$ connected in sequence. In this example, checking each pair of connected services independently may not be enough. Let us address two situations illustrated in Figure 6.

When the join point is the Dataport, the dependency between services is driven by **Rule 1**. The output dataport of $FService_i$, which is connected to the input dataport of $FService_{i+1}$, has to be compatible for $i \in 1...n$. In this case, the reasoning applies on a pair of services independently from the others. When the join point is the Functional Interface, **Rule 2** defined in Section 4 requires that the exchange protocol associated to $FService_i$ must be compatible with the ones of $FService_{i+1}$ for $i \in 1...n$. This requires the following checks to be made: *i)* FM_{ep1} and FM_{ep2} are consistent and also that *ii)* FM_{ep2} and FM_{ep3} are consistent. Let us now explain why it is necessary to reason globally on the entire sequence for this case. If we apply the same strategy as in Figure 5, we obtain:

$FM_{ep'2} \;=\; FM_{ep1} \oplus_{\cap} FM_{ep2} \quad (a)$

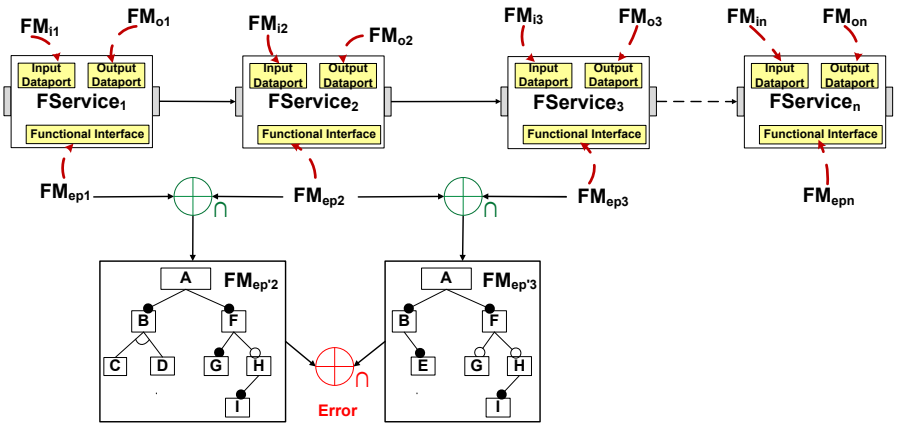


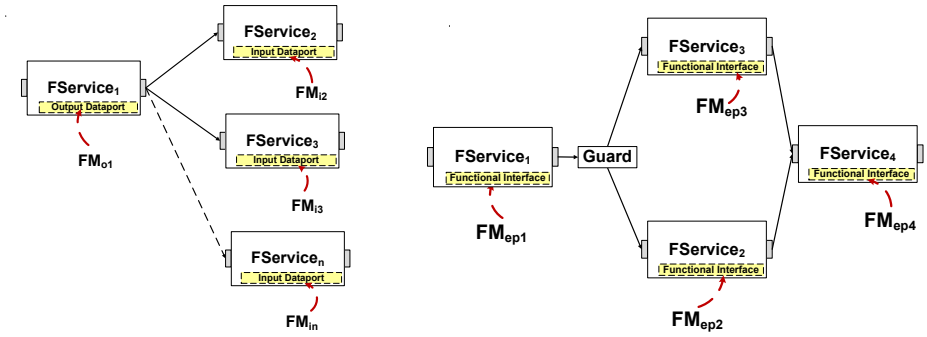
Fig. 6. Sequence of services

$FM_{ep'3} = FM_{ep2} \oplus_\cap FM_{ep3} \quad (b)$

However, the composition (a) has a side effect: some features of FM_{ep1} or FM_{ep2} may no longer be available. It is then possible that some features FM_{ep2} may not be involved in composition (b). Both compositions are therefore dependent on each other and should be addressed as a whole. Not following the above technique leads to an error, as shown in the bottom part of Figure 6: FM_{ep1} and FM_{ep2} are consistent as well as FM_{ep2} and FM_{ep3}, but the results of the two compositions are not compatible. We can generalize and state that: A sequence of $FService_1$, $FService_2$, ..., $FService_n$ is consistent according to a concern if and only if $SCR = ((FM_{ep1} \oplus_\cap FM_{ep2}) \oplus_\cap (FM_{ep2} \oplus_\cap FM_{ep3}) \oplus_\cap \ldots \oplus_\cap (FM_{ep(n-1)} \oplus_\cap FM_{epn})) \neq nil$, nil being the empty FM.

The merge operator properties M_1 and M_2 defined in Section 4 rely on the intersection or union of sets of configurations. In set theory, for the operations of intersection and union, associative, commutative and idempotent laws notably hold. The expression SCR can thus be simplified as follows:

$SCR = (FM_{ep1} \oplus_\cap FM_{ep2} \oplus_\cap FM_{ep3} \ldots \oplus_\cap FM_{epn}) \neq nil$



(a) Concurrency execution (b) Condition in the workflow

Fig. 7. Other Workflow Constructs

Concurrency. When two services are concurrently executed, the same situation occurs and it may not be sufficient to reason on pairs of services independently.

In Figure 7(a), $FService_1$ is connected to $FService_2$ and $FService_3$ which are concurrently executed. The medical image output supported by $FService_1$ is described with FM_{o1} which is attached to Output Dataport. The medical image input supported by $FService_2$ (resp. $FService_3$) is described with FM_{i2} (resp. FM_{i3}) which is attached to Input Dataport. We are considering that the medical image of $FService_1$ is transmitted to $FService_2$ and $FService_3$ (an output Dataport of $FService_1$ is connected to an input Dataport of $FService_2$ and an input Dataport of $FService_3$). In this case, the **Rule 1** applies but, as in the previous example, it is not sufficient to independently check each pair of services. Ensuring the satisfiability of the following formula is not sufficient:

$FM_{o1} \oplus_{\cap} FM_{i2} \neq nil \wedge FM_{o1} \oplus_{\cap} FM_{i3} \neq nil$

since the restrictions on the set of configurations of FM_{o1}, due to the merge of FM_{o1} and FM_{i2}, are not considered when composing FM_{o1} and FM_{i3}. As a result, the following relation must hold:

$FM_{o1} \oplus_{\cap} FM_{i2} \oplus_{\cap} FM_{i3} \neq nil$

It can be extended to n concurrent services as follows:

$FM_{o1} \oplus_{\cap} FM_{i2} \oplus_{\cap} FM_{i3} \ldots \oplus_{\cap} FM_{in} \neq nil$

Condition. When a condition is present in a workflow, different execution paths can be followed. This impacts the way (variable concerns of) services are dependent and thus consistency checking must be adapted. In Figure 7(b), the connection between services in the workflow means that $FService_1$ is executed, followed by an if-then-else condition: If the condition is true (resp. false), then $FService_2$ (resp. $FService_3$) is executed. In addition, $FService_2$ and $FService_3$ are connected to $FService_4$. The **Rule 2** applies between $FService_1$ and $FService_2$, $FService_1$ and $FService_3$, $FService_2$ and $FService_4$, as well as $FService_3$ and $FService_4$. There are two alternative paths (mutually exclusive) considering the execution flow: *i)* the execution of $FService_1$, then $FService_2$ and $FService_4$ or *ii)* the execution of $FService_1$, then $FService_3$ and $FService_4$. As a result, the following relation must hold:

$P_1 = (FM_{ep1} \oplus_{\cap} FM_{ep2} \oplus_{\cap} FM_{ep4}) \neq nil \wedge P_2 = (FM_{ep1} \oplus_{\cap} FM_{ep3} \oplus_{\cap} FM_{ep4}) \neq nil$

We propose to compute these restrictions as new FMs associated to each service. The new FM, $FM_{ep'1}$, associated to $FService_1$ is the union of the two paths in terms of sets of configuration: $FM_{ep'1} = P_1 \oplus_{\cup} P_2$. Then, the new FM, $FM_{ep'4}$, associated to $FService_4$ is also the union of the two paths in terms of sets of configuration: $FM_{ep'4} = P_1 \oplus_{\cup} P_2$. Finally, new FMs associated to $FService_2$ (resp. $FService_3$) are $FM_{ep'2} = P_1$ and $FM_{ep'3} = P_2$.

6 Assessment

Benefits and Strengths. If the variability manipulated by the user leads to some inconsistency but is considered to be more important than the workflow structure, the user has to correct the workflow itself. Using our approach, such inconsistencies can be systematically detected and several correction strategies can be applied. The separation of concerns provides the ability to precisely *locate the source of errors and to give information to assist users* in correcting the workflow. Hence, users can identify which specific services assembled in the workflow are causing inconsistency. In this case, a straightforward strategy is to

choose another service. Another solution is to identify and correct inadequate concerns, either by relaxing some variability description of services or by configuring differently some services (e.g., choosing a feature instead of another in a Xor-group). Another option is to detect and solve the shimming problem [17] by introducing intermediary workflow processes, called *shims*, that act as adapters between otherwise incorrectly wired services. The implementation of shims can solely focus on the inadequate concerns previously detected.

Collaborative and distributive development can also be implemented, e.g., several grid and medical imaging experts can *independently* and *incrementally* configure services and associated concerns with respect to their know-how. The merge operator deals with synchronizing choices and guarantees their coherence at each step.

Properties of the merge operator can then be exploited. The various compositions of FMs may be performed in any order because of the associativity property of the merge operator. Heuristics, such as merging larger FMs first, can thus be planned to detect an earlier source of errors. The idempotent and commutative properties can reduce the number of merge calls: In Figure 6, for n services sequentially executed, there are $n-1$ calls before simplification instead of $2*n-1$. The merge between FMs contributes to decrease the number of remaining variability choices.

An additional property of the merge in intersection mode is as follows: The number of features of the resulting FM is lesser than or equal to the number of features commonly shared by input FMs. This property can dramatically reduce the set of configurations to be considered by the user during workflow configuration. As a result, the *amount of time and effort needed during the configuration process can be reduced*. For instance, let us consider that the average of the number of features of each FM FM_{o1}, FM_{i2}, FM_{i3} and FM_{in} of Figure 7(a) is 30 and the number of features commonly shared by FMs is 20. After applying the merge operator, the new computed FM has a number of features which is necessary lesser than or equal to 20. As a result, a user only has to consider less than 20 features instead of $30*4=120$ features. Such benefits can also be observed for other workflows.

Current Limits and Threats. Currently we do not handle constraints between FMs whether they are internal or between several FMs. This is useful when concerns related to FMs are not independent, e.g., the QoS provided by a medical imaging service can be dependent on the kind of input images manipulated. More generally, the *feature interaction* problem is still an open and hard research challenge [14]. Constraints between FMs and feature interactions are threats to incremental and modular development, as well as to independent reasoning on FMs. They may cancel out some of the benefits presented above.

In the current proposal, we make the assumption that FMs to be merged have the same granularity, e.g., they share the same hierarchical structure. Given the open nature of the grid and the autonomy of the data and service providers, users may want to *align* concepts (features) and/or to negotiate some parts of FMs that are not present in another like in the viewpoints approach.

7 Related Work

Feature models. A few other approaches use multiple FMs during the SPL development. In [18], separate FMs are used to model decisions taken by different stakeholders or suppliers. The authors recognize the need to compose and merge FMs during multi-stage and multi-step configuration process, but do not achieve it. In [19], several FMs are used to separate feature descriptions related to requirements, problem world context and software specifications. Constraints then inter-relate features of FMs. Metzger et al. proposed a formal approach for separating PL variability (e.g., economical-oriented variability) and software variability, thereby enabling automatic analysis [20]. The two kinds of variability can be considered as concerns of an SPL. Previous contributions do not consider FMs or concerns that are sharing some features. This can happen when concerns along the same dimension interact, when multiple perspectives on a concern needs to be managed or when SPLs are composed with SPLs. A few works [10, 21, 22, 23] suggested the use of a merge operator: Our proposal goes further in this direction and clarifies the semantics of the merge and, most importantly, shows how this operator can be used in practice. In [24], an algorithm is designed to compute the kind of relations between two FMs. We have shown that reasoning on more than two FMs can happen for some constructs of the workflow and then, why the merge operator is required. In [25], the configuration process is represented as a workflow and different stakeholders are configuring the same FM. The first difference with our work is that the term workflow used in the approach does not refer to a processing pipeline, but to the activities completed during configuration. The second difference is that only a single FM is considered during the whole configuration process.

Multiple SPLs. In our case study, a medical imaging service can be seen as an SPL provided by different researchers or scientific teams. The entire workflow is then a multiple SPL in which different SPLs are composed. In many domains, organizations or architectures, the need to "shift from variation to composition" and to support multiple SPLs (also called product populations) is more and more patent [26]. Van der Storm considered not only variability at the level of one software product, but also each variable component as an entry-point for a certain software product (obtained through component composition) [27]. Hartmann and Trew dealt with multiple SPLs and identified several compositional issues in the context of software supply chains. They notably recognized that "merging FMs, especially when they are overlapping, requires a significant engineering activity" [23]. They did not provide a set of operators, a semantics nor a mechanism to automate this task. Reiser and Weber proposed to use multi-level feature trees consisting of a tree of FMs in which the parent model serves as a reference FM for its children [28]. Their purpose is mostly to cope with large diagrams and large-scale organizations, rather than different concerns.

Service composition. A large amount of work exists in (automatic) service composition (e.g. see [29]). To the best of our knowledge, there is no specific approach combining separation of concerns while managing variability in the same kind of context. In [30], AO4BPEL promotes a well-modularized specification of concerns and dynamic strategy for web service composition. Our work focuses on how to ensure in a processing chain, at design time, consistency between concerns with respect to variability. Work in [31] focused on how to map a FM

to a business process model described in BPEL; each feature of a FM corresponds to a business process. The motivation of our work is rather to describe the variability within a process; we also consider that the processing chain is fixed.

8 Conclusion and Future Work

Creating workflows from many different kinds of highly parameterized services is a cumbersome and error-prone task as important variabilities have to be managed by the user. In this paper, we have presented an approach that organizes services as a product line architecture and that uses feature models (FMs) to structure necessary information in terms of service variabilities. In the proposed approach, a family of services is defined as a set of concerns which exhibit variability, each being represented with one or several FMs. To reason on these artifacts, we rely on several related metamodels, reifying services, their dependencies in workflows and the join point related concepts. To enable the multiple composition of the concerns while taking variability into account, we have proposed a set of composition operators. Using these operators, we have defined consistency rules that enable the reasoning about the compatibility between families of connected services. Moreover, the consistency checking process makes it possible to automatically propagate variability choices into the whole workflow and thus to assist the user in selecting tailor-made services.

Future work aims at tackling current restrictions: *i)* handling inter- or intra-constraints between FMs in the composition process; *ii)* providing mechanisms to enable users to align FMs. The building of a large SPL dedicated to medical imaging services on the grid has already started. The services are part of a service-oriented architecture in which data-intensive workflows are built to conduct numerous computations on very large sets of images [4, 5]. The construction of such an SPL gives us an opportunity to obtain validation elements and feedback on the approach.

References

1. Taylor, I., Deelman, E., Gannon, D., Shields, M.: Workflows for e-Science. Springer-Verlag (2007)
2. Vigder, M., Vinson, N.G., Singer, J., Stewart, D., Mews, K.: Supporting scientists' everyday work: Automating scientific workflows. IEEE Software **25** (2008) 52–58
3. McPhillips, T., Bowers, S., Zinn, D., Ludäscher, B.: Scientific workflow design for mere mortals. Future Generation Computer Systems **25**(5) (May 2009) 541–551
4. Acher, M., Collet, P., Lahire, P.: Issues in Managing Variability of Medical Imaging Grid Services. In Olabarriaga, S., Lingrand, D., Montagnat, J., eds.: MICCAI-Grid, New York, NY, USA (2008) 10
5. Acher, M., Collet, P., Lahire, P., Montagnat, J.: Imaging Services on the Grid as a Product Line: Requirements and Architecture. In: Service-Oriented Architectures and Software Product Lines (SOAPL'08) workshop at SPLC'08, IEEE (2008)
6. Pohl, K., Böckle, G., van der Linden, F.J.: Software Product Line Engineering: Foundations, Principles and Techniques. Springer-Verlag (2005)
7. Acher, M., Collet, P., Lahire, P., France, R.: Composing Feature Models. In: 2nd International Conference on Software Language Engineering (SLE'09). Volume 5969 of LNCS. (2009) 62–81
8. Czarnecki, K., Eisenecker, U.: Generative Programming: Methods, Tools, and Applications. Addison-Wesley (2000)

9. Batory, D.S.: Feature models, grammars, and propositional formulas. In Obbink, J.H., Pohl, K., eds.: SPLC'05. LNCS, Springer (2005) 7–20
10. Schobbens, P.Y., Heymans, P., Trigaux, J.C., Bontemps, Y.: Generic semantics of feature diagrams. Comput. Netw. **51**(2) (2007) 456–479
11. Germain, C., Breton, V., et al.: XIX. In: Grid Analysis of Radiological Data. Handbook of Research on Computational Grid Technologies for Life Sciences, Biomedicine and Healthcare (2009) 30
12. Foster, I., Kesselman, C., Nick, J., Tuecke, S.: The Physiology of the Grid: An Open Grid Services Architecture for Distributed Systems Integration. Technical report, Open Grid Service Infrastructure WG, GGF (June 2002)
13. Deelstra, S., Sinnema, M., Bosch, J.: Product derivation in software product families: a case study. Journal of Systems and Software **74**(2) (2005) 173–194
14. Apel, S., Kästner, C.: An overview of feature-oriented software development. Journal of Object Technology (JOT) **8**(5) (July/August 2009) 49–84
15. `http://modalis.polytech.unice.fr/softwares/manvarwor`:
16. Muller, P.A., Fleurey, F., Jézéquel, J.M.: Weaving executability into object-oriented meta-languages. In: MODELS'05. LNCS, Springer (2005) 264–278
17. Lin, C., Lu, S., Fei, X., Pai, D., Hua, J.: A task abstraction and mapping approach to the shimming problem in scientific workflows. In: SCC '09: International Conference on Services Computing, IEEE (2009) 284–291
18. Czarnecki, K., Helsen, S., Eisenecker, U.: Staged Configuration through Specialization and Multilevel Configuration of Feature Models. Software Process: Improvement and Practice **10**(2) (2005) 143–169
19. Tun, T.T., Boucher, Q., Classen, A., Hubaux, A., Heymans, P.: Relating requirements and feature configurations: A systematic approach. In: SPLC'09, IEEE Computer Society (2009) 201–210
20. Metzger, A., Pohl, K., Heymans, P., Schobbens, P.Y., Saval, G.: Disambiguating the documentation of variability in software product lines: A separation of concerns, formalization and automated analysis. In: RE '07. (2007) 243–253
21. Alves, V., Gheyi, R., Massoni, T., Kulesza, U., Borba, P., Lucena, C.: Refactoring product lines. In: GPCE'06, ACM (2006) 201–210
22. Segura, S., Benavides, D., Ruiz-Cortés, A., Trinidad, P.: Automated merging of feature models using graph transformations. Post-proceedings of the Second Summer School on GTTSE **5235** (2008) 489–505
23. Hartmann, H., Trew, T.: Using feature diagrams with context variability to model multiple product lines for software supply chains. In: SPLC'08, IEEE (2008) 12–21
24. Thüm, T., Batory, D., Kästner, C.: Reasoning about edits to feature models. In: ICSE'09, IEEE Computer Society (2009)
25. Hubaux, A., Classen, A., Heymans, P.: Formal modelling of feature configuration workflows. In: SPLC'09, IEEE Computer Society (2009) 221–230
26. Ommering, R.C.v., Bosch, J.: Widening the scope of software product lines - from variation to composition. In: SPLC 2, Springer-Verlag (2002) 328–347
27. van der Storm, T.: Variability and component composition. In: Software Reuse: Methods, Techniques and Tools. LNCS (2004) 157–166
28. Reiser, M.O., Weber, M.: Multi-level feature trees: A pragmatic approach to managing highly complex product families. Requir. Eng. **12**(2) (2007) 57–75
29. Dustdar, S., Schreiner, W.: A survey on web services composition. Int. J. Web Grid Serv. **1**(1) (2005) 1–30
30. Charfi, A., Mezini, M.: AO4BPEL: an aspect-oriented extension to BPEL. World Wide Web **10**(3) (2007) 309–344
31. Schnieders, A., Puhlmann, F.: Variability modeling and product derivation in E-Business process families. In: Technologies for Business Information Systems. Springer (2007) 63–74